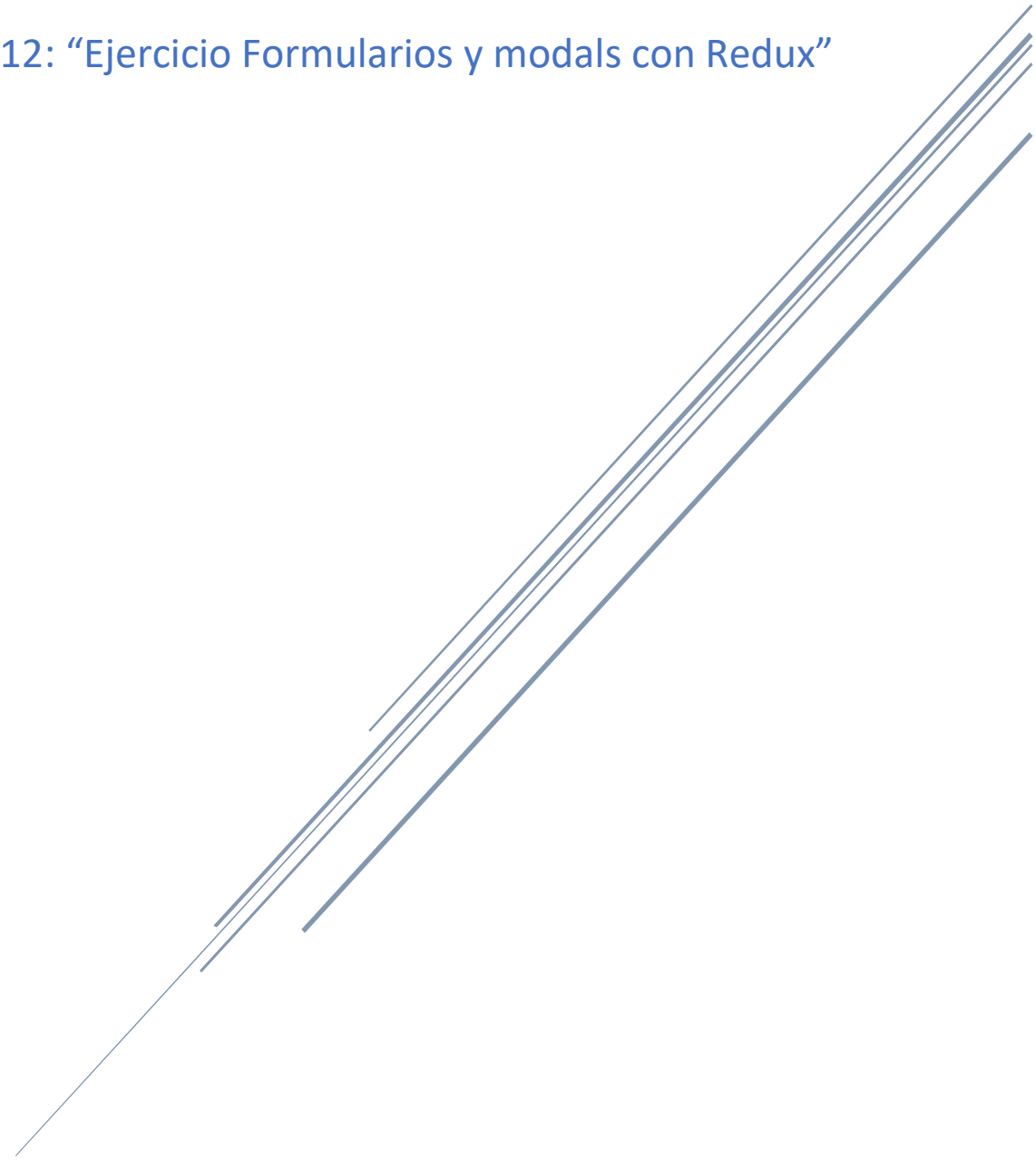
upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

APP GAZTAROA

Commit 12: “Ejercicio Formularios y modals con Redux”



Naiara Platero

Ejercicio Formularios y modals con Redux

En este segundo ejercicio no guiado, se va a crear un formulario para realizar comentarios en cada excursión.

1. Formulario con modal

Lo primero es añadir el botón que será un pencil de Font Awesome, gracias a Icon de React Native utilizado en commits anteriores.

```
<Icon style = {styles.icon}
                raised
                reverse
                name={ 'pencil' }
                type='font-awesome'
                color={colorGaztaroaOscuro}
                onPress={() => props.onClick()}
                />
```

A continuación, se crea el *Modal*, utilizando el elemento Rating de React Native para generar las estrellas, para mostrar el rating se utiliza *showRating*, para el formulario de autor y comentario se utiliza dos inputs con dos iconos, el icono *user* y el icono *comment*. Además, se añaden dos botones el botón *Enviar* y el botón *Cancelar*.

```
<Modal animationType = {"slide"} transparent = {false}
                visible = {this.state.showModal}
                onDismiss = {() => {this.toggleModal()}}
                onRequestClose = {() => {this.toggleModal()}}>
                <View style = {styles.modal}>
                <Rating
                  showRating
                  name="hover-feedback"
                  onFinishRating={rating => {console.log(rating); this.se
tState({ valoracion: rating })}}

                />
                <View style= {styles.input}>
                  <Input
                  placeholder={this.state.autor}
                  leftIcon={{ type: 'font-awesome', name: 'user'}}
                  onChangeText={value => this.setState({ autor: value })}
                  />
                  <Input
                  placeholder={this.state.comentario}
                  leftIcon={{ type: 'font-awesome', name: 'comment'}}
                  onChangeText={value => this.setState({ comentario: valu
e })}
                  />
                  <Button
```

```
                        onPress = {() =>{this.gestionarComentario(excursi
onId,this.state.valoracion,this.state.autor,this.state.comentario); this.
resetForm();}}
                        color={colorGaztaroaOscuro}
                        title="ENVIAR"
                        />
                  <Button
                        onPress = {() =>{this.toggleModal(); this.resetFo
rm();}}
                        color={colorGaztaroaOscuro}
                        title="CANCELAR"
                      />
                  </View>
               </View>
            </Modal>
```

Se crea la función, *toogleModal()* que será la responsable de invertir la variable de estado *showModal* en función del estado del *Modal.* Con la función *resetForm(),* se reseteará el formulario al pulsar el botón Cancelar.

La función *gestionarComentario(),* se encargará de actualizar los comentarios en Redux al pulsar el botón *Enviar*.

```
gestionarComentario(excursionId, valoracion, autor, comentario) {
  this.props.postComentario(excursionId, valoracion, autor, comentario);
  this.toggleModal();
}
```

2. Actualizar comentarios en Redux

Se crean dos nuevos tipos de acciones: POST_COMENTARIO y ADD_COMENTARIO. Se añaden al *ActionTypes.js.*

```
export const POST_COMENTARIO= 'POST_COMENTARIO';
export const ADD_COMENTARIO = 'ADD_COMENTARIO';
```

Ahora se modifica el *ActionCreators.js* asociadas a las acciones anteriores:

```
export const postComentario = (excursionId, valoracion, autor, comentario
)  => (dispatch) => {
    var dia = new Date();


    setTimeout(() => {
        dispatch(addComentario(excursionId, valoracion, autor, comentario
, dia));
    }, 2000);
};
```

```
export const addComentario = (excursionId, valoracion, autor, comentario,
 dia) => ({
    type: ActionTypes.ADD_COMENTARIO,
    excursionId: excursionId,
    valoracion: valoracion,
    autor: autor,
    comentario: comentario,
    dia: dia,
});
```

La función *postComentario()* recibirá cuatro parámetros desde el componente *DetalleExcursion* y creará un quinto parámetro *dia* antes de despachar la acción a *addComentario()* quien, a su vez, lo trasladará al *reducer*:

- Parámetro 1: ***excursionId***
- Parámetro 2: ***valoracion***
- Parámetro 3: ***autor***
- Parámetro 4: ***comentario***
- Parámetro 5: ***día*** (se genera utilizando *new Date()*)

También añade un retardo de 2 segundos para simular el envío al servidor.

La función *addComentario()*, se encargará de añadir el comentario guardando los valores del nuevo comentario.

A continuación, será necesario modificar el reducer *comentarios.js* que concatenará el comentario al array de comentarios. Respecto a la id se genera tomando la longitud del array en ese momento y restando uno, así la id no coincidirá nunca.

```
 case ActionTypes.ADD_COMENTARIO:
     let id=state.comentarios.length-1;
     return {...state,errMess: null, comentarios:
      [...state.comentarios,
        {id: id, excursionId: action.excursionId, valoracion: action.valo
racion, autor: action.autor, comentario: action.comentario,dia: action.di
a}
      ]
    };
```

Se modifica el componente *DetalleExcursion.js*, se crea el método mapDispatchToProps para poder lanzar la función Thunk que hemos creado en ActionCreators.js:

```
const mapDispatchToProps = dispatch => ({
    postFavorito: (excursionId) => dispatch(postFavorito(excursionId)),
    postComentario: (excursionId, valoracion, autor, comentario) => dispa
tch(postComentario(excursionId, valoracion, autor, comentario)),
})
```

Finalmente la función *gestionarComentario(),* hace uso de la función *postComentario():*

```
gestionarComentario(excursionId, valoracion, autor, comentario) {
  console.log(JSON.stringify(this.state));
  this.props.postComentario(excursionId, valoracion, autor, comentario);
  this.toggleModal();
}
```

Finalmente, la aplicación quedaría así: